Data Mining

Datenströme

Johannes Zschache
Wintersemester 2019

Abteilung Datenbanken, Universität Leipzig
http://dbs.uni-leipzig.de

Übersicht

Hochdimensionale Daten

- Clustering
- Dimensions-reduktion
- Empfehlungs-systeme
- Assoziations-regeln
- Locality Sensitive Hashing
- Supervised ML

Graphdaten

- Community Detection
- PageRank
- Web Spam

Datenströme

- Windowing
- Filtern
- Momente
- Web Advertising

Inhaltsverzeichnis

Literatur: Kapitel 4 aus „Mining of Massive Datasets“: http://www.mmds.org

Datenströme

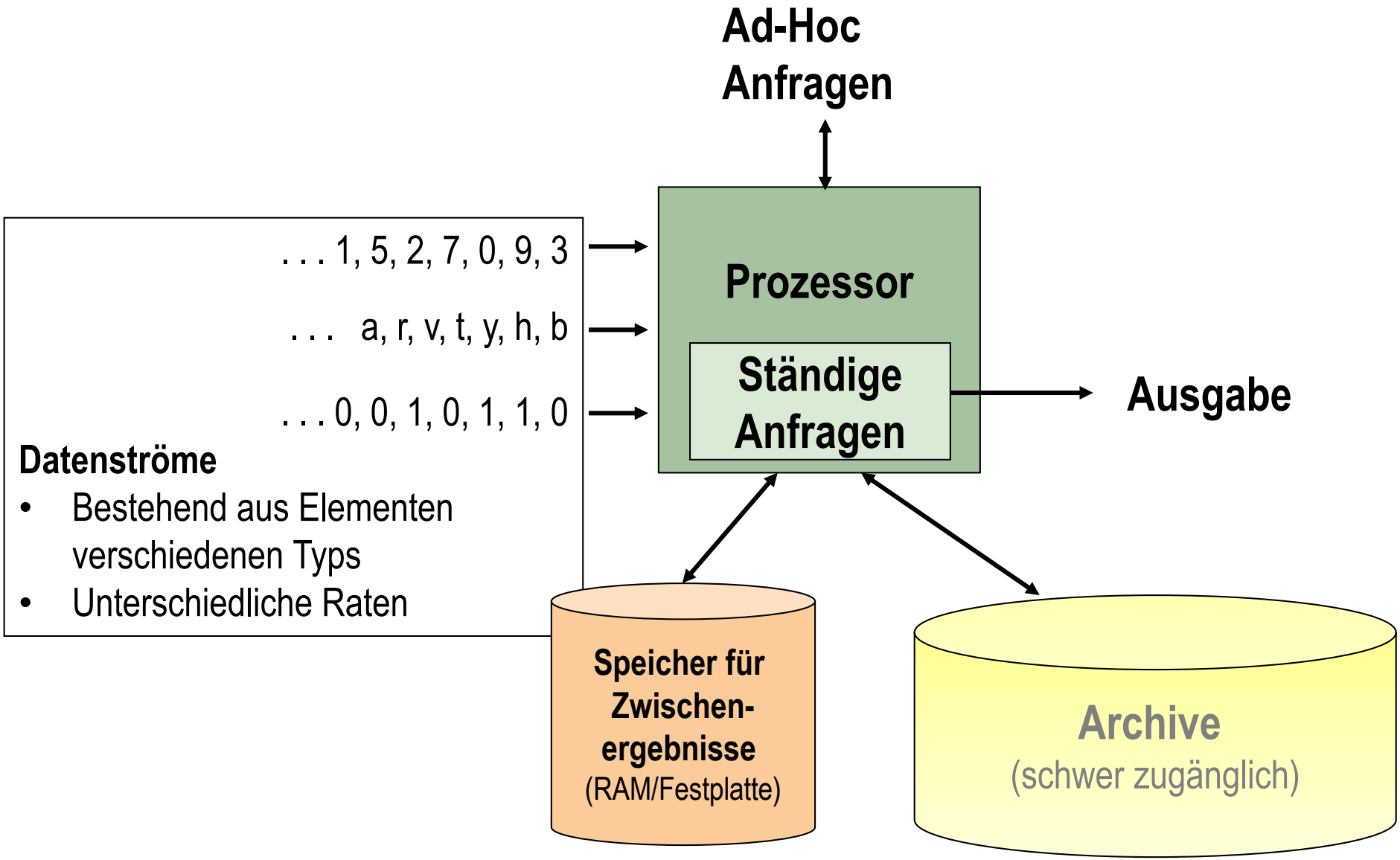
- In vielen Situationen liegt der Datensatz zum Zeitpunkt der Analyse nicht vollständig vor

- *Datenstrom* bedeutet, dass Daten **unbegrenzt** sind und die Rate der Eingabe **nicht kontrollierbar** ist

- **Annahmen**
  - Daten treten als „unbegrenzter Strom“ mit einer **sehr großen Rate** über einen oder mehrere Eingänge auf
  - Zugängliche *Speicherung* der Daten (auf Festplatte) nicht möglich
  - Daten werden **sofort verarbeitet** und danach verworfen

- **Frage**: Wie sind kritische Berechnungen über den Strom mit einer begrenzten Menge an Speicher möglich?

Datenstrommodel



Beispiele

- **Sensordaten**
  - Milliarden Sensoren senden Daten an zentrale Einheit
  - z.B. Wasserstand/Temperatur in Flüssen/Seen/Meeren
- **Anfragen an Suchmaschine**
  - Welche Anfragen werden heute häufiger gestellt, als gestern?
  - z.B. Anfragen wie „Symptome Grippe“ können auf Grippewelle hinweisen
- **Clickstream**
  - Welche Wikipedia-Seiten wurden in der letzten Stunde besonders häufig besucht?
  - Welche Nachrichten/Themen sind aktuell?
- **News-Feed auf Sozialen Medien**
- **IP/Telefon-Daten**
  - Informationen für optimales Routing der Packete / Denial-of-Service-Attacken
  - Erstellung von Abrechnungen für Kunden/Unternehmen
- Online **Machine Learning** (z.B. Stochastic Gradient Descent)

Inhaltsverzeichnis

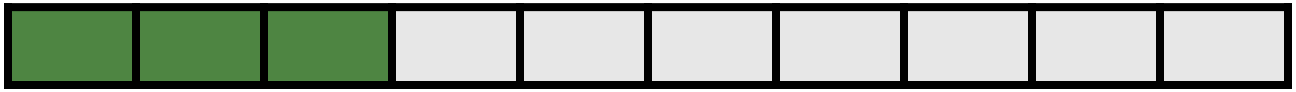
Literatur: Kapitel 4 aus „Mining of Massive Datasets“: http://www.mmds.org

Ziehen einer Stichprobe

- Auswahl und Speicherung einer **repräsentativen Teilmenge** der Daten
- Anfragen auf Stichprobe anstatt gesamten Daten
- Zwei Arten
  1. **Fester Anteil** des Datenstroms: z.B. 1 von 10 Elementen, d.h. die Stichprobe wächst kontinuierlich mit der Zeit (mit dem Datenstrom)
  2. **Feste Anzahl** an Elemente: z.B. 10.000 Elemente, d.h. die Stichprobe hat eine feste maximale Größe

Stichprobe mit festen Anteil

- Elemente: (ID, Zeitpunkt, Attribut 1, Attribut 2, …)
- **Implementierung** über **Hashfunktion:**
  - Wähle eine/mehrere Komponenten der Elemente als Schlüssel, z.B. ID, Zeitpunkt oder ein Attribut
  - Für einen Anteil von $\frac{a}{b}$, verwende *b* Buckets und speichere ein Element, falls der Hash-Wert des Schlüssels in einen der ersten *a* Buckets fällt
  - ***z.B. für eine 30%-Stichprobe:***



- **Wahl des Schlüssels hängt von Anwendung/Frage ab**
- **Beispiel**: Anfragen an Suchmaschine
  - Elemente: (ID, Zeitpunkt, Nutzer, Anfrage)
  - ***Frage: Wie oft stellen die Nutzer die gleiche Anfrage (im Durchschnitt)?***
  - Anteil: 10%

Stichprobe mit festen Anteil

- **Naive Lösung**: Hash-Funktion auf ID der Elemente
  - Auswahl von 10% der Elemente
  - **Problem**: Abweichung des Erwartungswerts der Stichprobe von tatsächlichem Wert
  - Beispiel:
    - Ein Nutzer stellt x einzigartige Anfrage und d Duplikate
    - **Exakte Antwort**: $\frac{d}{d+x}$ der Anfragen sind Duplikate
    - **ABER**: Erwartungswert der naiven Schätzung: $\frac{d}{19d+10x}$
    - In Stichprobe: Duplikate tauchen mit Wahrscheinlichkeit $\frac{1}{100}$ als Duplikat und Wahrscheinlichkeit $\frac{18}{100}$ als einzigartige Anfrage auf

- **Bessere Lösung**: Hash-Funktion auf Name/ID des Nutzers
  - Auswahl von 10% der Nutzer
  - Speicher alle Anfragen der Nutzer

Stichprobe fester Größe

- Stichprobe darf eine **Größe s** nicht überschreiten, z.B. aufgrund von Speicherbeschränkungen
- ***Erwünschte Eigenschaft***: Nachdem n Elemente eintrafen, ist jedes dieser Elemente mit Wahrscheinlichkeit $\frac{s}{n}$ Teil der Stichprobe
- Beispiel mit s = 2: a x c y z k c d e g…
  - Nach **n = 7**, alle 7 Elemente sind mit Wahrscheinlichkeit $\frac{2}{7}$ Teil der Stichprobe
  - Nach **n = 10**, alle 10 Elemente sind mit Wahrscheinlichkeit $\frac{2}{10}$ Teil der Stichprobe

Algorithmus (a.k.a. Reservoir Sampling)

- Speichere die ersten s Elemente des Datenstroms
- Angenommen es trifft das $n$te Element ein ($n > s$)
  - Mit Wahrscheinlichkeit ***s/n***, speichere das ***n***te Element
  - Falls das ***n***te Element gespeichert wurde, ersetze eins der vorhandenen s Elemente (zufällig ausgewählt nach Gleichverteilung) durch dieses Element

Beweis der erwünschten Eigenschaft

- **Vollständige Induktion**
- **IA:** Nachdem $n = s$ Elemente eintrafen, sind alle Elemente mit Wahrscheinlichkeit $\frac{s}{n} = \frac{s}{s} = 1$ in der Stichprobe
- **IS:** Angenommen die erwünschte Eigenschaft gilt nachdem n Elemente eintrafen. Für das Eintreffen des $(n+1)$ten Elements gilt:
  - Das $(n+1)$te Element ist mit Wahrscheinlichkeit $\frac{s}{n+1}$ in der Stichprobe
  - Jedes Elemente *aus vorherigen Stichprobe* bleibt darin mit Wahrscheinlichkeit

$$\left(1 - \frac{s}{n+1}\right) + \frac{s}{n+1} \cdot \frac{s-1}{s} = \frac{n}{n+1}$$

  - *Jedes* der ersten n Elemente ist in Stichprobe mit Wahrscheinlichkeit

$$\frac{s}{n} \cdot \frac{n}{n+1} = \frac{s}{n+1}$$

Inhaltsverzeichnis

Literatur: Kapitel 4 aus „Mining of Massive Datasets“: http://www.mmds.org

Anfragen mit Sliding Window

- Anfragen sind auf ein Sliding Window der Länge N, d.h. auf *die neuesten N Elemente*, beschränkt
- Beispiel mit N = 6:

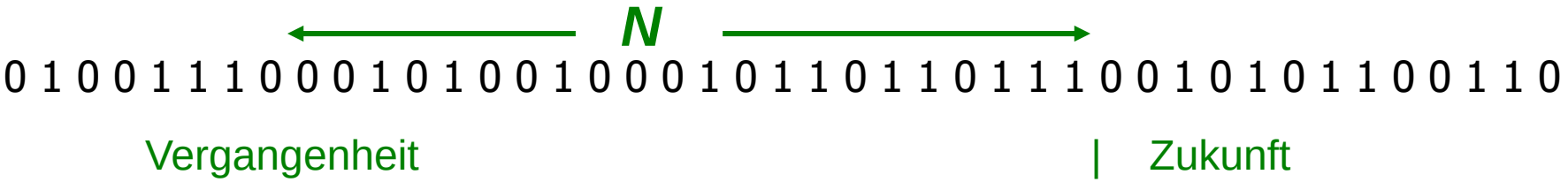
- **Problem:** unzureichender Speicherplatz für Sliding Windows, falls
  - N ist zu groß
  - zu viele Datenströme

- Beispiel: Online Händler
  - Ein 0/1-Datenstrom pro Produkt
  - Information, ob das Produkt in der n-ten Transaktion verkauft wurde
  - Anfrage: Wie oft wurde ein Produkt in den letzten k Transaktionen verkauft?

Bits Zählen

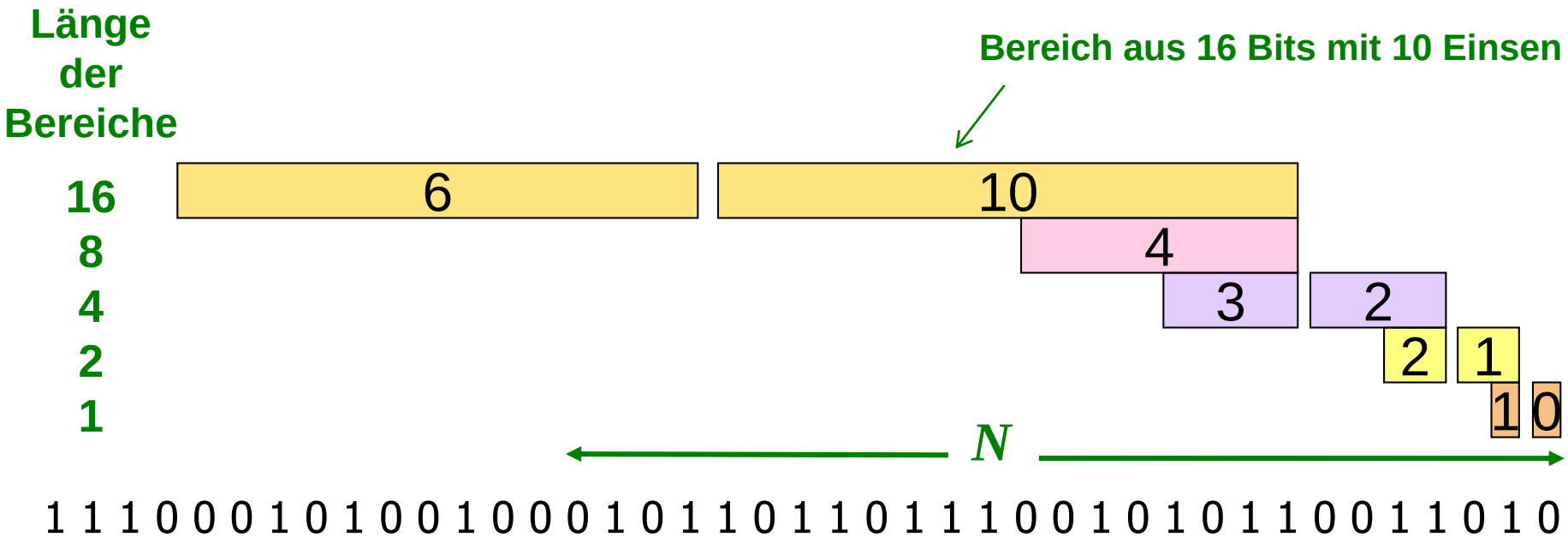
- Gegeben ist ein 0/1-Strom, ein Sliding Window der Länge N und Anfragen der Form:

  Wie viele Einsen sind unter den letzten k Bits (k ≤ N)?



- ***Exakte Antwort*** nur durch **Speichern aller N Bits** möglich
- ***Approximative Antwort*** durch Speichern aggregierter Daten: Anzahl der Einsen in einem festen Bereich
- z.B. nur 2 Zähler und Annahme der Gleichverteilung (schlechte Lösung)
  - ***S***: Anzahl der Einsen seit Beginn
  - ***Z***: Anzahl der Nullen seit Beginn
  - **Anzahl der Einsen unter den letzten k Bits:** $k \cdot \frac{S}{S+Z}$

Exponentiell wachsende Bereiche

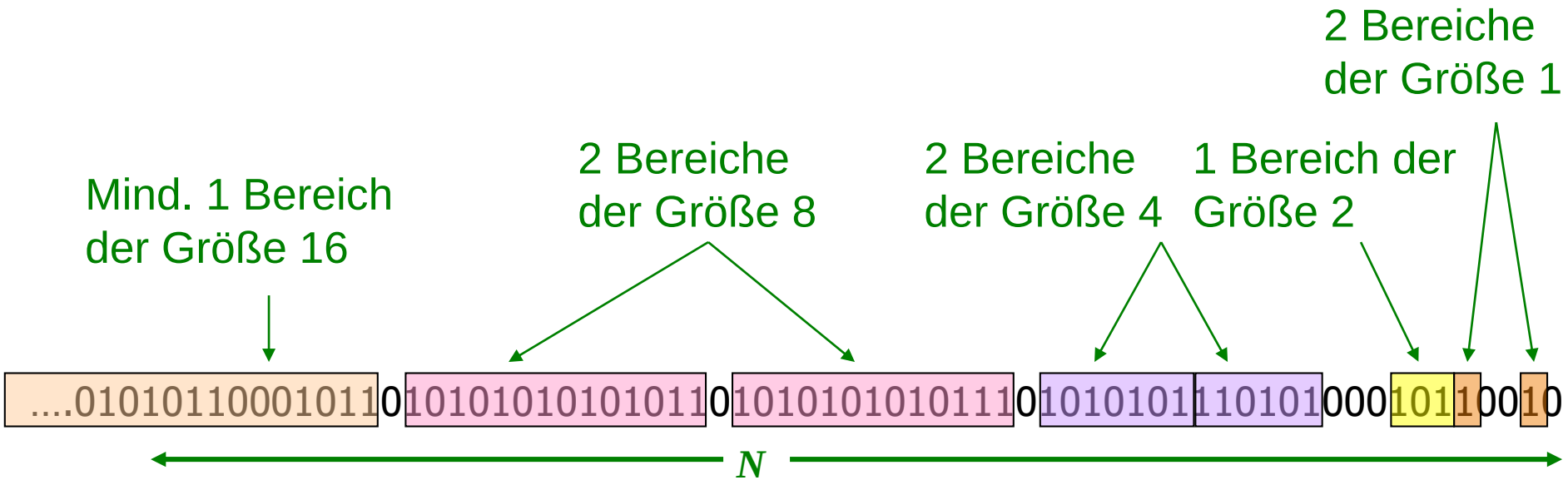


- **Vorteil**: **Speicherbedarf**
  - Anzahl der Bereiche: $\boldsymbol{O}(\log_2 \mathrm{N})$
  - Insgesamt: $\boldsymbol{O}(\log_2^2 \mathrm{N})$ Bits (Speichern der Anzahl der Einsen pro Bereich benötigt maximal $\log_2 N$ Bits)
- **Nachteil**: großer Fehler möglich, z.B. falls alle Einsen im ältesten Bereich

DGIM Methode

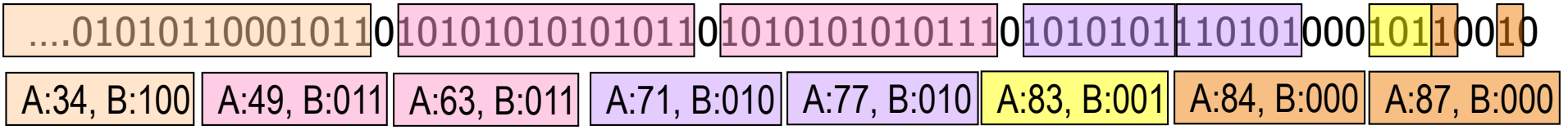
Lösung: **D**atar-**G**ionis-**I**ndyk-**M**otwani-Methode

Anstatt von Bereichen mit bestimmter Länge, Zusammenfassen von ***Bereichen mit bestimmter Anzahl an Einsen***



DGIM-Bereiche

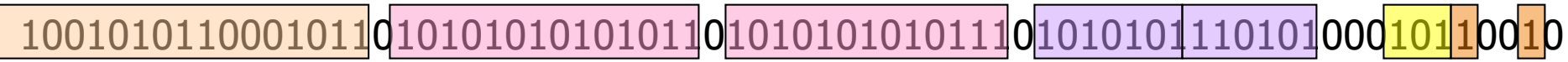
- Jedes Element (Bit) des Datenstroms hat einen *Zeitpunkt*: 1, 2, …
- Ein **DGIM-Bereich** besteht aus
  A. Dem Zeitpunkt des neuesten Eintrags
  B. Anzahl der Einsen im Bereich = die *Größe* des DGIM-Bereichs
- Regeln:
  - Jedes 1-Bit fällt in genau einen Bereich
  - Das rechte Ende eines Bereichs enthält eine Eins
  - Die Größe eines Bereichs (Anzahl der Einsen) muss eine **Zweierpotenz** sein
    → Speicherung der Größe benötigt nur $\log_2 \log_2 \mathrm{N}$ Bits
  - Von dem neuesten zu älteren Bereichen ist dessen Größe monoton *steigend*
  - Bereiche verschwinden, wenn dessen Zeitpunkt außerhalb des Sliding Windows liegt
  - Es gibt entweder einen oder zwei Bereiche einer Größe → $\boldsymbol{O}(\log_2 \mathrm{N})$ Bereiche



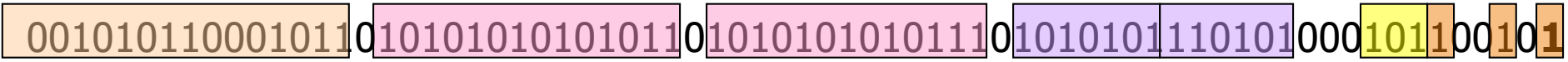
Aktualisierung der DGIM-Bereiche

- Ankommen eines neuen Bit
- Evtl. verschwindet ältester Bereich (falls sein Zeitpunkt außerhalb des Sliding Window liegt)
- Falls neuer Bit eine Null: keine weiteren Änderungen notwendig
- Falls neuer Bit eine Eins:
  1. Erstelle neuen Bereich der Größe 1 und aktuellem Zeitpunkt
  2. Falls nun 3 Bereich der Größe 1: Kombiniere die älteren beiden Bereich zu einem Bereich der Größe 2
  3. Falls nun 3 Bereich der Größe 2: Kombiniere die älteren beiden Bereich zu einem Bereich der Größe 4
  4. usw.
- Maximal $\log \mathrm{N}$ Schritte: $\boldsymbol{O}(\log_2 \mathrm{N})$ Zeitkomplexität

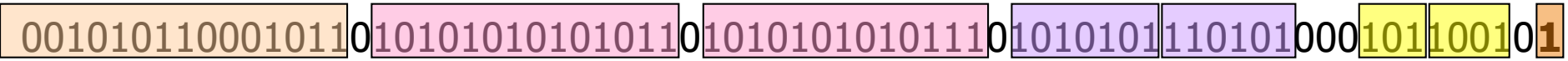
DGIM: Beispiel



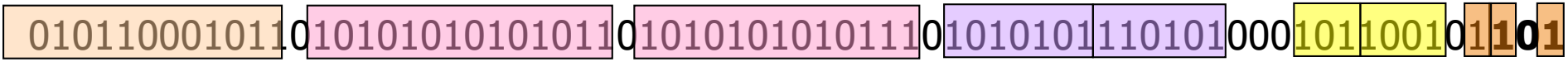
Ankommen eines 1-Bits



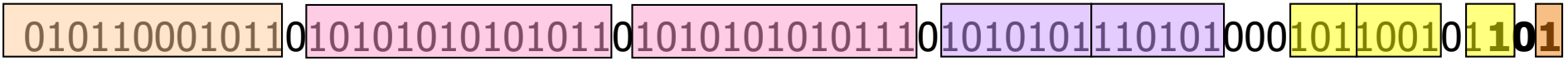
Vereinigung zweier Bereiche



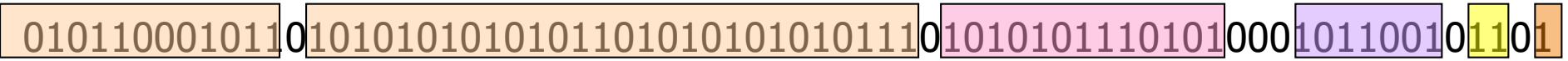
Ankommen weiterer Bits



Vereinigung der Bereiche



Vereinigung der Bereiche …

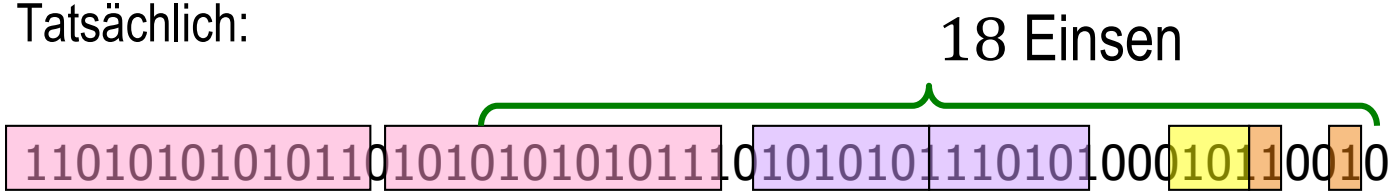


DGIM: Anfragen

Wie viele Einsen sind unter den letzten k = 35 Bits?

A:49, B:011 | A:63, B:011 | A:71, B:010 | A:77, B:010 | A:83, B:001 | A:84, B:000 | A:87, B:000

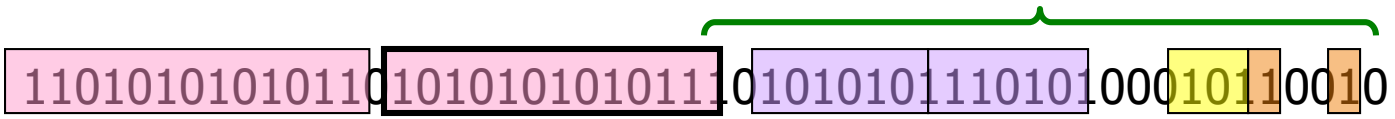
- Suche den ältesten Bereich B dessen Zeitpunkt noch innerhalb der neuesten k Bits liegt
  - Sei aktueller Zeitpunkt $t = 88$
  - Ältester Bereich mit $A > t - k = 53$: A:63, B:011
- Schätze die Anzahl der Einsen durch:
  - Summe der Größen aller Bereiche, die neuer als B sind
  - Plus die Hälfte der Größe von B
  - Beispiel: $1 + 1 + 2 + 4 + 4 + \frac{8}{2} = 16$
  - Tatsächlich:



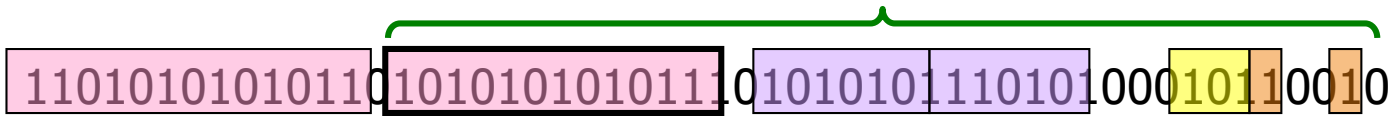
DGIM: Fehler

- Fehler: **max. 50 %**
  - Sei 2^r die Größe von B
  - Tatsächliche Anzahl der Einsen $\geq$ der Anzahl der Einsen in den vorherigen Bereichen plus Eins $\geq 1 + 2 + 4 + \cdots + 2^{r-1} + 1 = 2^r$

  - Man addiert also maximal die Hälfte der wahren Anzahl



  - Bzw. es fehlt maximal die Hälfte von B



Exponentially Decaying Window

- Nachteile des Sliding Window
  - Nur möglich bei relativ wenig 0/1-Strömen, da Bereiche gespeichert werden
  - Weniger geeignet bei dem Zählen von Paaren/Mengen, da ein separater Strom pro Paar/Menge nötig

- Alternative: **Exponentially Decaying Window** (EDW)
  - Anstatt Festlegung auf die letzten N Elemente, Aggregation über gesamten Datenstrom und ***höhere Gewichtung der neuesten Elemente***
  - Seien $\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots$ die 0/1-Elemente eines Elements $\boldsymbol{x}$ und c eine Konstante (z.B. 10^{-6} oder 10^{-9}), **Maß für die Häufigkeit** von $\boldsymbol{x}$ zum Zeitpunkt t:

$$\boldsymbol{S_t(x) = \sum_{i=1}^{t} a_i (1-c)^{t-i}}$$

  - Einfache Aktualisierung: Bei Ankunft eines neuen Elements $\boldsymbol{x}$
    - $\boldsymbol{S_{t+1}(x) \leftarrow 1 + (1-c) S_t(x)}$
    - $\boldsymbol{S_{t+1}(y) \leftarrow (1-c) S_t(y)}$ für alle $\boldsymbol{y \neq x}$

Exponentially Decaying Window

- **Speicherung von nur einem Zähler pro Element/Strom**
- Außerdem: Löschen von Zählern $\boldsymbol{S_t(x)}$, welche kleiner als ein Schwellenwert $\boldsymbol{s}$ sind
- Eigenschaft:

$$\sum_{\boldsymbol{x}} \boldsymbol{S_t(x)} = \sum_{\mathbf{i=1}}^{\boldsymbol{t}} \boldsymbol{(1-c)^{t-i}} < \sum_{i=0}^{\infty} (1-c)^i = \frac{1}{1-(1-c)} = \frac{1}{c}$$

- Für jeden Schwellenwert $\boldsymbol{s}$ (z.B. $\boldsymbol{s = 2}$), kann es maximal $1/cs$ Elemente mit $\boldsymbol{S_t(x) \geq s}$ geben
- Die Anzahl der Elemente für die ein Wert $\boldsymbol{S_t(x)}$ gespeichert wird, ist begrenzt durch $\boldsymbol{s}$

Inhaltsverzeichnis

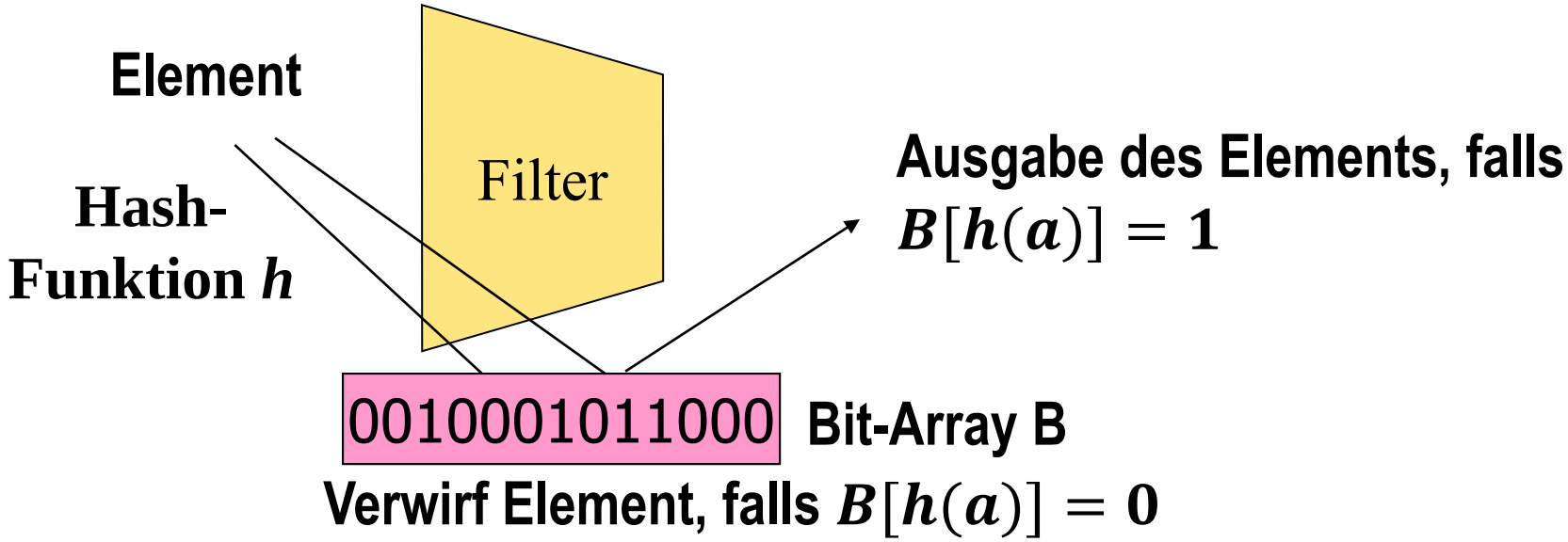
Literatur: Kapitel 4 aus „Mining of Massive Datasets“: http://www.mmds.org

Filter auf Datenströmen

- Leicht umsetzbare Filter:
  - Attribut Alter > 18
  - Attribut Kategorie == A
- **Schwieriger**: Attribut ist Element einer Menge S und **S ist sehr groß**
- *Problem*: S passt nicht in Hauptspeicher oder mehrere Millionen Filter, die zusammen nicht in Speicher passen
- ***Beispiele*:**
  - **E-Mail-Spam-Filter**: S ist die Menge von 1 Milliarde E-Mail-Adressen
  - **Publish-Subscribe-Systeme**: z.B. Newsletter, die aus vielen Nachrichten/Blogeinträgen zusammengestellt und an Abonnenten mit unterschiedlichen Interessen gesendet werden
  - **Content-Filter**: Jede Anzeige sollte nur einmal pro Nutzer geschaltet werden

Filter über eine einzelne Hash-Funktion

- Array B aus n Bits und Hash-Funktion h, welche den Wertebereich von S gleichmäßig auf die Menge $\{0,1, \ldots, n-1\}$ abbildet
- Initialisierung:
  - Setze alle Bits auf 0
  - Für jedes $s \in S$, setze $B[h(s)] = 1$
- Gegeben ein Element eines Datenstroms mit Attribut a: Wähle Element genau dann aus, wenn $B[h(a)] = 1$



Filter über eine einzelne Hash-Funktion

Sei S eine Menge von m Elementen und B eine Array aus n Bits ($m < n$)

- *Ungefähr* ein Anteil $\frac{m}{n}$ aller Bits sind auf Eins gesetzt
- Falls ein Attribut a zu S gehört, dann ist $B[h(a)] = 1$ garantiert → **Keine False Negatives**
- Falls ein Attribut b *nicht* zu S gehört, wird es mit einer Wahrscheinlichkeit von ungefähr $\mathbf{1 - e^{-m/n}}$ einem Bit mit einer Eins zugeordnet → **False Positive**
- Wahrscheinlichkeit eines False Positive:

$$1 - \left(1 - \frac{1}{n}\right)^{m} = 1 - \left(1 - \frac{1}{n}\right)^{n\frac{m}{n}} \approx 1 - e^{-m/n}$$

- **Beispiel**: 1 Milliarde E-Mail-Adressen und 1 GB Array, d.h. $m = 10^9$ und $n = 8 \cdot 10^9$: $1 - e^{-1/8} = 0.1175$

Bloom Filter

- Verwende k unabhängige Hash-Funktionen $h_1, h_2, \ldots, h_k$
- Initialisierung:
  - Setze alle Bits von B auf 0
  - Für jedes $s \in S$ setze $B[h_i(s)] = 1$ für alle $i = 1, .., k$
- Für ein Element aus dem Datenstrom mit Attribut a:
  - Falls $B[h_i(a)] = 1$ **für alle** $i = 1, .., k$, gib Element aus
  - Ansonsten: Verwirf Element
- Wahrscheinlichkeit, dass ein gegebenes Bit auf Eins gesetzt wurde:

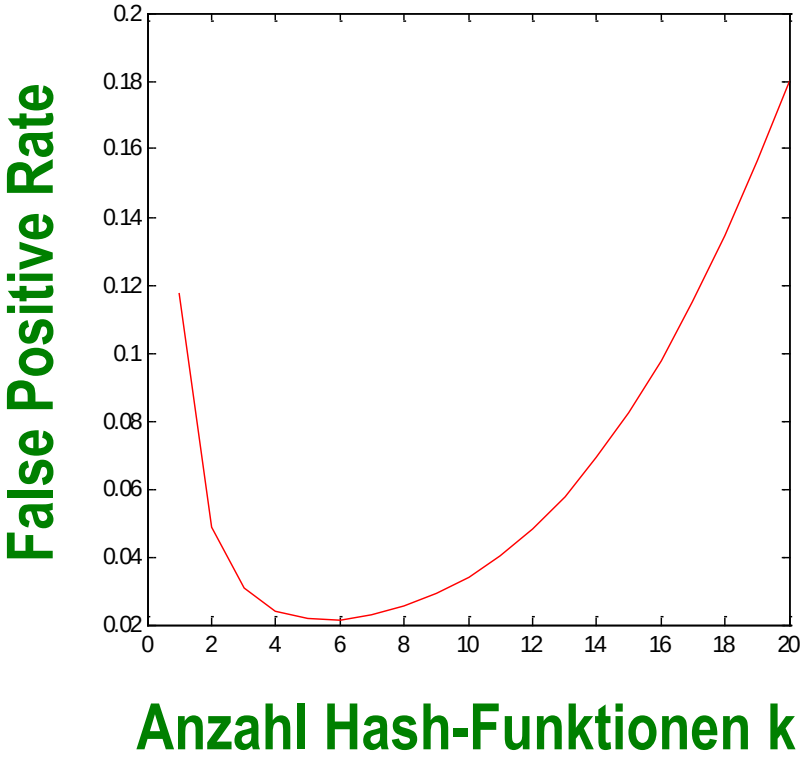
$$1 - \left(1 - \frac{1}{n}\right)^{km} \approx 1 - e^{-km/n}$$

- Wahrscheinlichkeit eines False Positive:

$$\left(1 - e^{-km/n}\right)^k$$

Bloom Filter: Optimierung

- **Beispiel**: 1 Milliarde E-Mail-Adressen und 1 GB Array
- d.h. $m = 10^9$ und $n = 8 \cdot 10^9$
  - $k = 1: 1 - e^{-1/8} \approx 0.12$
  - $k = 2: \left(1 - e^{-1/4}\right)^2 \approx 0.05$
  - $k = 3: \left(1 - e^{-3/8}\right)^3 \approx 0.03$
  - $k = 4: \left(1 - e^{-1/2}\right)^4 \approx 0.02$
  - …



- **Optimum:** $k_{opt} = \frac{n}{m} \ln 2$
- Beispiel: $k_{opt} = 8 \ln 2 \approx 6$
- Fehler für $k = 6$: $\left(1 - e^{-6/8}\right)^6 \approx 0.004$

Inhaltsverzeichnis

Literatur: Kapitel 4 aus „Mining of Massive Datasets“: http://www.mmds.org

Anzahl eindeutiger Elemente

- Frage: Anzahl der eindeutigen Elemente, die bisher auftraten

- Beispiele
  - Wie viele verschiedene Nutzer haben eine Webseite im letzten Monat besucht?
  - Wie viele verschiedene Webseiten wurden von einem Nutzer aufgerufen?
  - Wie viele verschiedene Produkte wurden letzte Woche verkauft?

- Welches Vorgehen eignet sich, wenn die bisher gesehenen Elemente nicht in den Hauptspeicher passen bzw. tausende Datenströme gleichzeitig verarbeitet werden?

- **Ziel: Schätzung der Anzahl mit möglichst geringem Fehler**

Flajolet-Martin Algorithmus

- Hash-Funktion h, die eine Menge von m verschiedenen Elementen auf eine Sequenz von mind. $\log_2 \mathrm{m}$ Bits abbildet
  - z.B. $h(e) = 10100$
- Für jedes Element e, sei $r(e)$ die Anzahl der hinteren Nullen von $h(e)$
  - z.B. falls $h(e) = 10100$, dann ist $r(e) = 2$

- Speichere R als das Maximum von $r(e)$ über alle bisherigen Elemente
- **Schätzer für die Anzahl der eindeutigen Element**: $\mathbf{2^R}$

- *Intuition*:
  - Je mehr verschiedene Elemente beobachtet werden (je größer m), desto mehr unterschiedliche Hash-Werte werden berechnet
  - Je mehr unterschiedliche Hash-Werte berechnet werden, desto wahrscheinlicher ist das Auftreten eines sehr seltenen Hash-Wertes

Flajolet-Martin Schätzer: Begründung

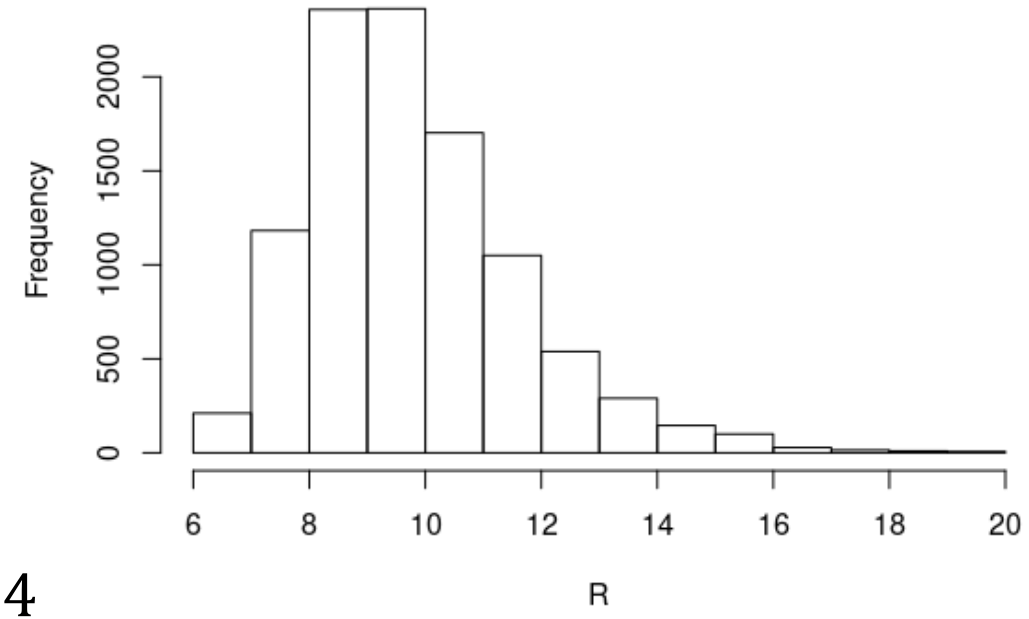
- Die Werte von $h(e)$ haben gleiche Wahrscheinlichkeit
- Die Wahrscheinlichkeit einen Hash-Wert mit r hinteren Nullen zu berechnen is 2^{-r}
  - Ungefähr 50% der Werte haben die Form ***0
  - Ungefähr 25% der Wert haben die Form **00
  - …
- Wahrscheinlichkeit *mindestens einen* Hash-Wert mit r hinteren Nullen durch die Eingabe von m verschiedenen Elementen zu beobachten ist

$$1 - (1 - 2^{-r})^m = 1 - (1 - 2^{-r})^{2^r m 2^{-r}} \approx 1 - e^{-\frac{m}{2^r}}$$

- Beispiel: $m = 1000$
  - $r = 9$: $1 - e^{-\frac{m}{2^r}} = 0.85$
  - $\boldsymbol{r = 10}$: $1 - e^{-\frac{m}{2^r}} = 0.62$
  - $r = 11$: $1 - e^{-\frac{m}{2^r}} = 0.38$

Flajolet-Martin Schätzer: Probleme

- **Simulation** mit $m = 1000$ und Bit-Array der Größe 20; 10 000 Wiederholungen
- Falls $R = 10$ (24% der Fälle), dann Schätzung von m durch $2^R = 1024$



- Aber extreme **Überschätzung**, falls $R > 10$ (38% der Fälle) durch z.B. $2^{11} = 2048$ (17%) oder $2^{12} = 4096$ (9%)
- Lösung: Verwendung mehrerer Hash-Funktionen
  - Mittelwert: anfällig gegenüber Ausreißern (Simulation: 5243)
  - Median: nur Zweierpotenzen (Simulation: 1024)
  - Alternative: Unterteilung in kleine Gruppen von Hash-Funktionen und Mittelwert der Mediane (Simulation mit Gruppen aus 50 Hash-Funktionen: 1048)

Inhaltsverzeichnis

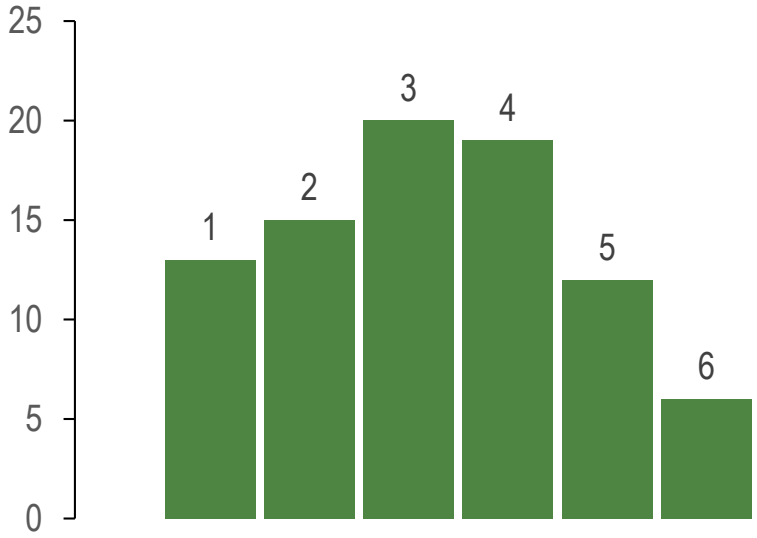
Literatur: Kapitel 4 aus „Mining of Massive Datasets“: http://www.mmds.org

Häufigkeitsverteilung

- Gegeben: Datenstrom aus *natürlichen Zahlen*
- Sei m_i die Anzahl des Vorkommens von $i \in \mathbb{N}$ im Datenstrom
- **Häufigkeitsverteilung**

$i \in A$	1	2	3	4	5	…
m_i	m_1	m_2	m_2	m_3	m_5	…

- Darstellung als Histogramm:



Momente

- Betrachtung des Falls: Alle m_i passen **nicht** in den Hauptspeicher
- Charakterisierung der Häufigkeitsverteilung über deren **Momente**
- Sei $A \subset \mathbb{N}$ die endliche Menge der tatsächlich vorkommenden Zahlen
- Das **k-te Moment** ist definiert als

$$\frac{1}{|A|}\sum_{i \in A}(m_i)^k$$

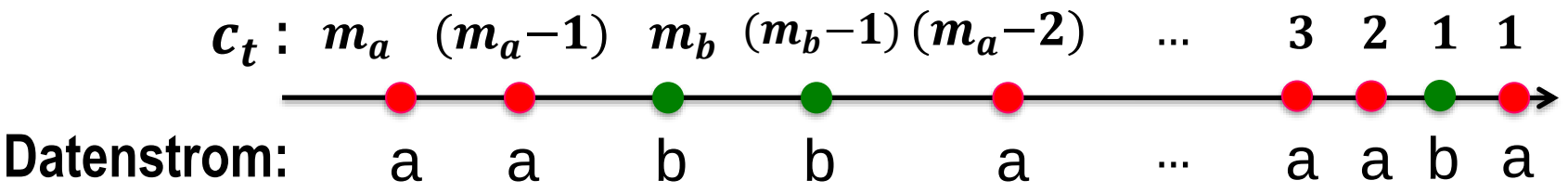
- 1te Moment = Durchschnittliche Häufigkeit der Zahlen
- 2te Moment = Maß der Ungleichheit
  - z.B. Datenstrom aus 100 Elementen mit $A = \{1, 2, .., 11\}$
  - m_i: 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9 → 2tes Moment: 82.7
  - m_i: 90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1 → 2tes Moment: = 737.3
- Anmerkung: Empirische Varianz: (2te Moment) – (1te Moment)^2

Alon-Matias-Szegedy Algorithmus

- **Schätzung des 2-ten Moments**
- Speicherung einer großen (durch den Hauptspeicher begrenzten) Anzahl an Variablen $X_1, X_2, \ldots, X_l$ mit $X_j = \left(X_j.elem, X_j.val\right)$
- Sei n die Länge des Datenstroms (eigentlich unbekannt)
- Für jede Variable X_j
  - Wähle (gleichmäßig) zufällig einen Zeitpunkt t ($t \leq n$)
  - $X_j.elem = i$, wobei $i \in A$ das Element des Datenstroms zum Zeitpunkt t
  - $X_j.val = c_t$, wobei c_t die Anzahl des Vorkommens von i **ab Zeitpunkt *t***
  - Sei $f\left(X_j\right) = n(2 \cdot X_j.val - 1)$
- **Schätzer des 2-ten Moments**:

$$S = \frac{1}{|A|l} \sum_{j=1}^{l} f\left(X_j\right)$$

Alon-Matias-Szegedy Algorithmus



$$E[f(X)] = \sum_{t=1}^{n} \frac{1}{n} \cdot n(2c_t - 1)$$

$$= \sum_{i \in A} \sum_{j=1}^{m_i} (2j-1) = \sum_{i \in A} \left(2 \sum_{j=1}^{m_i} j - \sum_{j=1}^{m_i} 1 \right)$$

$$= \sum_{i \in A} \left(2 \frac{m_i(m_i+1)}{2} - m_i \right) = \sum_{i \in A} (m_i)^2$$

$\rightarrow \frac{1}{|A|} f(X)$ ist unverzerrter Schätzer des 2-ten Moments

Alon-Matias-Szegedy Algorithmus

- Momente höherer Ordnung
  - Für das k-te Moment:

$$f(X) = n\,(c^k - (c-1)^k), \text{ wobei } X.val = c$$

  - Beispiel: $k = 3$

$$f(X) = n\,(3c^2 - 3c + 1)$$

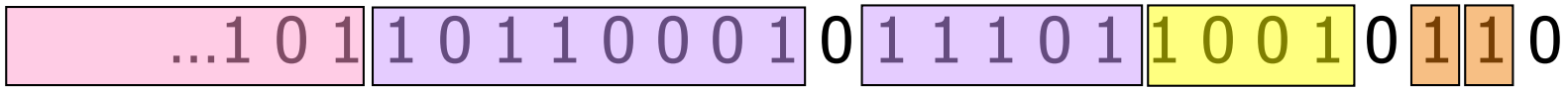
- In Praxis: Anstatt Mittelwert über alle Variablen, Median über Mittelwerte von kleineren Gruppen von Variablen
- Unbegrenzte Daten (n unbekannt): Lösung über *Reservoir Sampling*
  - Stichprobengröße l
  - Wähle ersten l Zeitpunkte für die Variablen
  - Wenn das n-te Element ($n > l$) auftritt, wähle es mit Wahrscheinlichkeit $\frac{l}{n}$
  - Falls das n-te Element als neue Variable gewählt wird, entferne eine existierende Variable mit Wahrscheinlichkeit $\frac{1}{l}$

Inhaltsverzeichnis

Literatur: Kapitel 4 aus „Mining of Massive Datasets“: http://www.mmds.org

Übung 1

Angenommen Sie sehen für den abgebildeten Datenstrom nur die gekennzeichneten DGIM-Bereichen:



a) Wie lautet der DGIM-Schätzer für die Anzahl der Einsen unter den letzten k Bits für
   i. $k = 5$
   ii. $k = 15$

b) Wie stark ist die Abweichung zu der tatsächlichen Anzahl an Einsen in beiden Fällen?

c) Wie verändern sich die DGIM-Bereiche wenn drei weitere Einsen hinzukommen?

Übung 1: Lösung

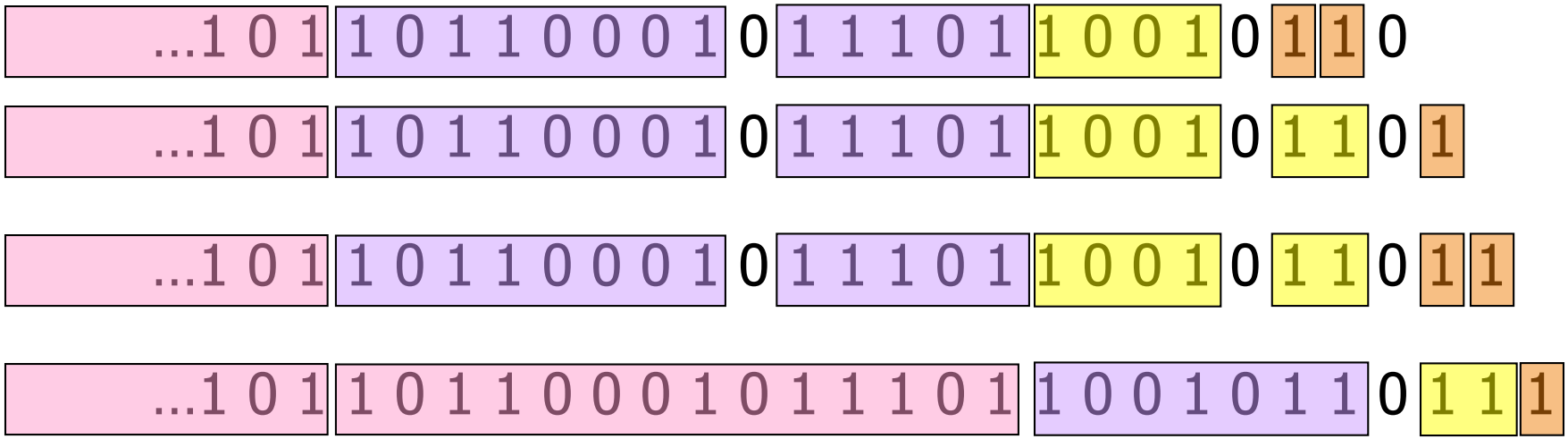
Angenommen Sie sehen für den abgebildeten Datenstrom nur die gekennzeichneten DGIM-Bereichen:

a) Wie lautet der DGIM-Schätzer für die Anzahl der Einsen unter den letzten k Bits für
   i. $k = 5$: **3**
   ii. $k = 15$: **10**

b) Wie stark ist die Abweichung zu der tatsächlichen Anzahl an Einsen?
   i. $k = 5$: **3** $\rightarrow$ **Fehler: 0**
   ii. $k = 15$: **9** $\rightarrow$ **Fehler: 1**

Übung 1: Lösung

c) Wie verändern sich die DGIM-Bereiche wenn drei weitere Einsen hinzukommen?



Übung 2

Gegeben ist ein Datenstrom aus ganzen Zahlen:

3 1 4 1 5 9 2 6 5

Schätzen Sie die Anzahl eindeutiger Elemente über den Flajolet-Martin Algorithmus! Verwenden Sie folgende Hash-Funktionen:

a) $h(x) = 2x + 1 \; mod \; 32$

b) $h(x) = 3x + 7 \; mod \; 32$

c) $h(x) = 4x \; mod \; 32$

Übung 2: Lösung

Gegeben ist ein Datenstrom aus ganzen Zahlen:

3 1 4 1 5 9 2 6 5

a) $h(x) = 2x + 1 \; mod \; 32$

- h(x): 7 3 9 3 11 19 5 13 11
- Anzahl hintere Nullen: 0 0 0 0 0 0 0 0 0
- $2^0 = 1$

b) $h(x) = 3x + 7 \; mod \; 32$

- h(x): 16 10 19 10 22 34 13 25 22
- Anzahl hintere Nullen: 4 1 0 1 1 1 0 0 1
- $2^4 = 16$

c) $h(x) = 4x \; mod \; 32$

- h(x): 12 4 16 4 20 36 8 24 20
- Anzahl hintere Nullen: 2 2 4 2 2 2 3 3 2
- $2^4 = 16$

Mittelwert: 11

Tatsächlich: 7

Übung 3

Gegeben ist ein Datenstrom aus ganzen Zahlen:

3 1 4 1 3 4 2 1 2

Berechnen Sie das 2-te Moment der Häufigkeitsverteilung über den Alon-Matias-Szegedy Algorithmus! Verwenden Sie, anstelle der zufälligen Auswahl, eine Variable X_j für jedes Element des Datenstroms!

Übung 3: Lösung

Gegeben ist ein Datenstrom aus ganzen Zahlen:

3 1 4 1 3 4 2 1 2

Berechnen Sie das 2-te Moment der Häufigkeitsverteilung über den Alon-Matias-Szegedy Algorithmus! Verwenden Sie, anstelle der zufälligen Auswahl, eine Variable X_j für jedes Element des Datenstroms!

- $X_i.val$: 2 3 2 2 1 1 2 1 1
- $f(X) = n(2\ X_i.val - 1)$
- $f(X_i.val)$: 27 45 27 27 9 9 27 9 9
- Mittelwert: 21
- 2-tes Moment: $21/4$